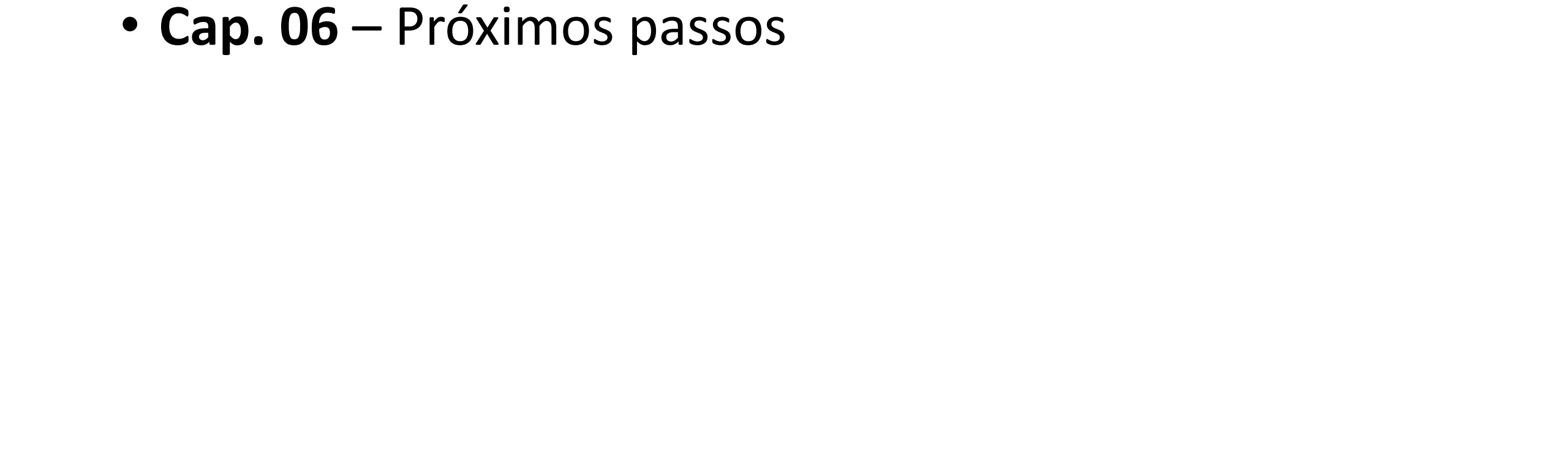
# NO PYTHON DE LA CASA

MANUAL DO PROFESSOR
PARA CONSTRUIR FUNÇÕES ROBUSTAS E PRECISAS

# INDICE





01

EL PROFESSOR

# MERGULHANDO NO MUNDO DAS FUNCOES

# O QUE SAO FUNCOES

Imaginem um programa como um grande roubo: as **funções** são as peças do plano, cada uma com um papel específico para que o código seja *organizado*, *eficiente* e *reutilizável*. Com elas, evitamos repetições e facilitamos a leitura, como se cada parte tivesse um título e um resumo. Trabalhar em equipe também fica mais fácil, dividindo as tarefas como se estivessem dividindo o “ouro" do cofre. E se algo der errado? As funções facilitam a vida! Resolvemos o problema em uma parte específica, sem afetar o resto do programa. Dominar as funções em **Python** é como se tornar um professor experiente: organizamos o código com maestria e construímos programas robustos!

**Veja o exemplo básico de uma Função a seguir:**

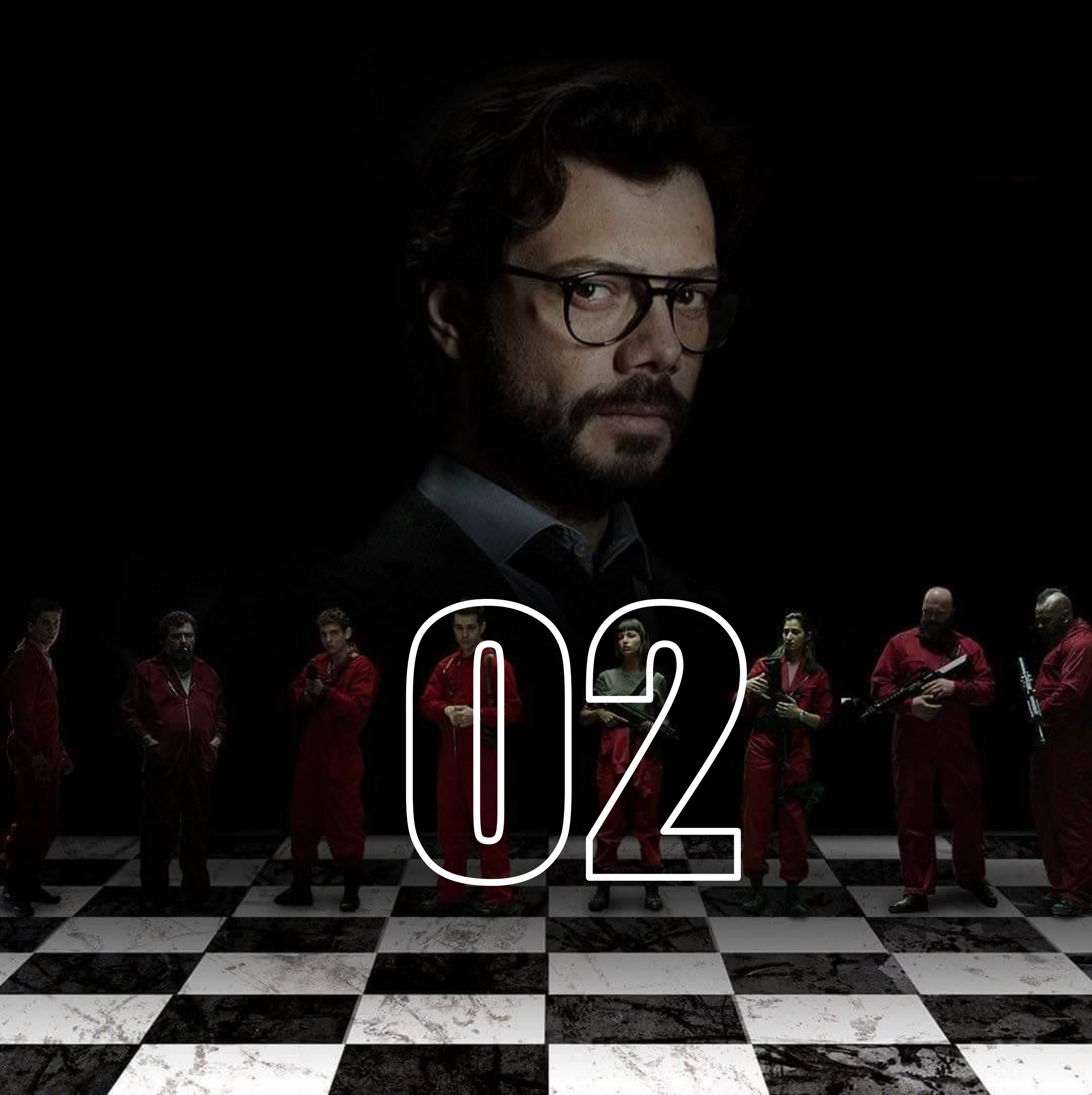
```python
def saudar(nome):
  print(f"Olá, {nome}! Seja bem-vindo(a)!")

# Testando a função
nome = "João"

saudar(nome)
```

```
Olá, João! Seja bem-vindo(a)!
```

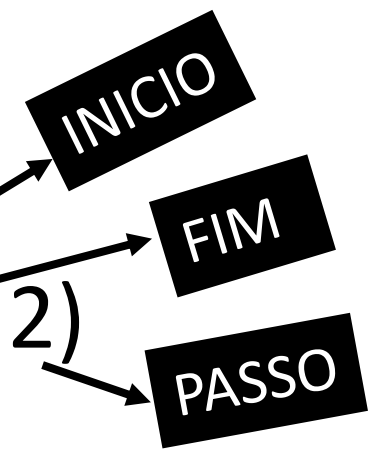
# ANATOMIA DE UMA FUNCAO



# ANATOMIA DE 1 FUNCAO

## A ESTRUTURA BÁSICA:

1. **Palavra-chave def:** É a marca registrada de uma função, como se fosse o início de uma nova aventura.
2. **Nome da função:** Identifica a função e deve ser descritivo. Ex: def calculo_notas.
3. **Parâmetros:** São os argumentos que a função recebe, como ferramentas para realizar seu trabalho. Ex: conte de 0 a 10 de 2 em 2. (0,10, 2)
4. **Corpo da função:** Contém as instruções que a função executa, como os passos para salvar o mundo.




```python
def somar(numero1, numero2):
    """Soma dois números e retorna o resultado."""
    soma = numero1 + numero2
    return soma

resultado = somar(10, 20)
print(f"Resultado da soma: {resultado}")  # Resultado da soma: 30
```

1 2 3 4

# ANATOMIA DE 1 FUNCAO

## DOMINANDO OS PARÂMETROS

**Parâmetros obrigatórios:** São aqueles que a função precisa para funcionar, como água e comida para um herói.

**Parâmetros opcionais:** Possuem valores padrão que podem ser usados se nenhum valor for especificado, como uma capa invisível.

**Parâmetros nomeados:** Permitem que você especifique qual argumento corresponde a cada parâmetro, como diferentes poderes para cada membro da equipe.

```python
def calcular_area(base, altura, tipo="retângulo"):
  """Calcula a área de uma forma geométrica."""
  if tipo == "retângulo":
    area = base * altura
  elif tipo == "triângulo":
    area = (base * altura) / 2
  else:
    print(f"Forma geométrica {tipo} não suportada.")
    return None

area_retangulo = calcular_area(5, 4)
area_triangulo = calcular_area(base=6, altura=3, tipo="triângulo")

print(f"Área do retângulo: {area_retangulo}")
# Área do retângulo: 20
print(f"Área do triângulo: {area_triangulo}")
# Área do triângulo: 9.0
```

# ANATOMIA DE 1 FUNCAO

## RETORNANDO VALORES:

**Funções podem retornar valores:** Como se estivessem entregando um presente para o usuário. E podem ser retornos simples ou múltiplos.

**Retorno simples:** Retorna um único valor, como a resposta para uma pergunta.
Ex: Imagine a função que calcula a soma de 2 números. Seu objetivo é retornar o resultado da soma para usar em outras partes do código.

```python
def somar(numero1, numero2):
    """Soma dois números e retorna o resultado."""
    soma = numero1 + numero2
    return soma

resultado = somar(10, 20)
print(f"Resultado da soma: {resultado}")
# Resultado da soma: 30
```

- A função somar recebe 2 números como argumentos (numero1 e numero2), realiza a soma e utiliza a palavra-chave **return** para **retornar o valor da soma** (soma).
- A variável resultado armazena o valor retornado pela função somar, que pode ser utilizado posteriormente no código.

# ANATOMIA DE 1 FUNCAO

## RETORNANDO VALORES:

**Retorno múltiplo:** Retorna vários valores de uma só vez, é como vários presentes em uma entrega só.
Ex: considere a função que precisa retornar mais de um valor, como a altura e a largura de um retângulo.

```python
def calcular_dimensoes_retangulo(base, altura):
  """Calcula a área e o perímetro de um retângulo
  e retorna ambos os valores."""
  area = base * altura
  perimetro = 2 * (base + altura)
  return area, perimetro

dimensoes = calcular_dimensoes_retangulo(5, 4)
area_retangulo, perimetro_retangulo = dimensoes

print(f"Área do retângulo: {area_retangulo}")
# Área do retângulo: 20
print(f"Perímetro do retângulo: {perimetro_retangulo}")
# Perímetro do retângulo: 18
```

- A função calcular_dimensoes_retangulo recebe a base e a altura como argumentos, calcula a área e o perímetro, e utiliza **return** para **retornar ambos os valores** como uma tupla (area, perimetro).
- As variáveis area_retangulo e perimetro_retangulo desempacotam a tupla **retornada** pela função, armazenando cada valor em uma variável separada.

# ANATOMIA DE 1 FUNCAO

## RETORNANDO VALORES:

**OBS**: Em algumas situações, Quando as Coisas Não Funcionam Conforme o Planejado, uma função pode precisar retornar um valor especial para indicar que algo deu errado. O valor **None** é utilizado para representar a ausência de um valor em Python.

```python
def abrir_porta(chave, codigo_secreto):
  """Abre a porta se a chave e o código secreto estiverem corretos.
  Retorna True em caso de sucesso, False caso contrário."""
  if chave == "chave123" and codigo_secreto == 456:
    print("Porta aberta!")
    return True
  else:
    print("Falha ao abrir a porta.")
    return None

porta_aberta = abrir_porta("chave123", 456)
if porta_aberta:
  print("Acesso autorizado!")
else:
  print("Acesso negado.")
```

•A função abrir_porta verifica se a chave e o código secreto estão corretos. Se estiverem, ela imprime uma mensagem, **retorna** True (indicando sucesso) e a porta é aberta.Caso contrário, a função imprime uma mensagem de falha, **retorna** None (indicando erro) e a porta permanece fechada.

03

# DESVENDANDO OS MISTÉRIOS DA FUNÇÃO LAMBDA

# FUNCOES LAMBDA

## Desvendando os mistérios da função lambda

As funções lambda, também conhecidas como funções anônimas, são ferramentas poderosas em Python que **permitem criar funções curtas e concisas em uma única linha de código**. Elas são ideais para situações onde você precisa de uma função simples para uma tarefa específica, sem a necessidade de criar uma função completa com nome e definição formal.

**Veja o exemplo básico de uma Função Lambda a seguir:**

```python
ordenar_por_crescente = lambda lista: sorted(lista)

numeros = [5, 2, 4, 1, 3]
numeros_ordenados = ordenar_por_crescente(numeros)
print(numeros_ordenados)  # Resultado: [1, 2, 3, 4, 5]
```

# FUNCOES LAMBDA

## Anatomia da Função Lambda:

A sintaxe de uma função lambda é bastante concisa e composta por três elementos principais:

1. **Palavra-chave** “lambda”: Esta palavra indica o início da definição da função lambda.
2. **Parâmetros**: Após a palavra-chave lambda, você pode definir os parâmetros da função (caso seja mais de um), separar por vírgulas.
3. **Expressão**: Após os dois pontos (:), vem a expressão que define o que a função lambda deve fazer. Essa expressão pode ser composta por operações matemáticas, comparações, chamadas de outras funções e até mesmo estruturas de controle de fluxo.

Veja o exemplo básico de uma Função Lambda a seguir:

```python
def filtrar_pares(lista):
  return list(filter(lambda x: x % 2 == 0, lista))
#                    1       2      3

numeros = [1, 2, 3, 4, 5, 6]
numeros_pares = filtrar_pares(numeros)
print(numeros_pares)  # Resultado: [2, 4, 6]
```

# 04

# DESBRAVANDO OS SEGREDOS DE ARGS E KWARGS

# ARGS

**Se seu objetivo é** Criar funções que podem receber **muitos argumentos** de diferentes formas. Muito provável que **seus inimigos sejam Confusão** e **código mal escrito**.

**Com args e kwargs, isso é resolvido!**

**O que é ARGS?**

São como **caixas mágicas** que armazenam os argumentos que você passa para suas funções.

```
def assalto_plano (tokio, rio, denver, oslo):
  dinheiro_roubado = tokio + rio + denver + oslo
  print (f"Dinheiro roubado: {dinheiro_roubado} milhões!")

assalto_plano (40, 30, 25, 15)  # Dinheiro roubado: 110 milhões!
```

- A função assalto_plano recebe **muitos membros da gangue** com args.
- Cada membro é um **argumento posicional**, como se fossem os membros do time.
- A função soma o dinheiro roubado por cada membro e revela o total!

# KWARGS

**kwargs:** Captura **argumentos nomeados**, como se cada um tivesse um poder especial!

```python
def preparar_explosivo(tipo="dinamite", quantidade=50, detonador="detonador remoto"):
  print(f"Nairobi prepara {quantidade} kg de {tipo} com {detonador}.")

preparar_explosivo(tipo="C4", quantidade=20)
# Nairobi prepara 20 kg de C4 com detonador remoto.
preparar_explosivo(detonador="fio detonador")
# Nairobi prepara 50 kg de dinamite com fio detonador.
```

- A função preparar_explosivo recebe o **tipo** e **quantidade** do explosivo como argumentos nomeados obrigatórios.
- O **detonador** é um argumento nomeado opcional com valor padrão.
- A função imprime as informações sobre o explosivo que Nairobi prepara.

**Lembrem-se:**

- kwargs permitem personalizar funções para se adaptar a diferentes situações.
- Nairobi usa kwargs para ser versátil e eficiente em qualquer tarefa!

05

# DOCUMENTANDO FUNCOES A CHAVE PARA A CLAREZA

# DOCUMENTANDO

## Com documentação adequada, suas funções se tornam armas poderosas!

**1. O Que É Documentação?**

A documentação é como um **manual de instruções** para suas funções. Ela explica o que a função faz, como usá-la e quais informações ela espera receber. É como se vocês estivessem deixando um mapa do tesouro para seus colegas.

**2. Como Documentar em Python:**

•Utilizem as **docstrings** (3 aspas duplas), que são como comentários especiais logo abaixo da definição da função. Ex: “””

•Comecem com um resumo da função, em **uma ou duas linhas**. Ex: “””a função calc, calcula a soma entre 2 números”””.

•Detalhem os argumentos da função, especificando seus tipos e valores padrão.

•Expliquem o que a função retorna, se for o caso.

•Usem exemplos para ilustrar como utilizar a função.

# DOCUMENTANDO

```python
def abrir_cofre(heist_leader, disguise="máscara de Dali", city="Madri"):
    """
    Abre o cofre usando o disfarce do líder do assalto.

    Argumento:
        heist_leader (str): Nome do líder do assalto.
        disguise (str, opcional): Disfarce usado pelo líder (padrão: "máscara de Dali").
        city (str, opcional): Cidade onde o cofre está localizado (padrão: "Madri").

    Retorno:
        bool: True se o cofre for aberto, False caso contrário.
    """
    # Código para abrir o cofre...

    if cofre_aberto:
        print(f"Cofre aberto por {heist_leader} em {city}!")
        return True
    else:
        print(f"Falha ao abrir o cofre em {city}!")
        return False

abrir_cofre("Tóquio")  # Abre o cofre com o disfarce padrão em Madri.
abrir_cofre("Rio", disguise="óculos escuros", city="Tóquio")  # Abre o cofre em Tóquio com óculos escuros.
```

Toda essa parte textual, explica o que a função faz.
Pode ser acionada pela função
**help**(nome_da função) em qualquer momento.

**Lembrem-se:**

- Documentação clara e concisa torna seu código mais fácil de entender e usar.
- É como se vocês estivessem deixando pistas para seus colegas encontrarem o sucesso.
- Pratiquem e documentem suas funções com cuidado para se tornarem mestres em Python!

# 06

# PROXIMOS PASSOS

LUIZBUSSATTO

# PROXIMOS PASSOS

Parabéns por ter dominado as funções em Python! Este é um passo fundamental na sua jornada como desenvolvedor. Para continuar evoluindo e se tornar um profissional cada vez mais experiente, siga estas dicas:

1. **Aprofunde seus conhecimentos em Python:** Domine as estruturas de controle: Aprenda a utilizar if, else, for, while e outras estruturas para controlar o fluxo do seu código de forma eficiente.
2. **Explore módulos e pacotes**: O Python oferece uma vasta biblioteca de módulos e pacotes prontos para diversos fins, como manipulação de dados, web scraping, análise de texto e muito mais.
3. **Desvende a Orientação a Objetos**: Crie classes, objetos e heranças para organizar seu código de forma modular e reutilizável, tornando seus projetos mais escaláveis e fáceis de manter.
4. **Domine o tratamento de exceções**: Aprenda a lidar com erros e falhas no seu código de forma robusta e elegante, garantindo a estabilidade e confiabilidade das suas aplicações.

# PROXIMOS PASSOS

## Pratique e Construa Projetos:

- **Comece com projetos simples:** Crie scripts para automatizar tarefas do dia a dia, como organizar arquivos, enviar emails ou realizar cálculos.
- **Participe de desafios de programação:** Diversas plataformas online oferecem desafios de programação que te ajudarão a aprimorar suas habilidades e solucionar problemas de forma criativa. Codewars.com é uma boa plataforma para isso.
- **Contribua para projetos open-source:** Colabore com projetos em código aberto no GitHub, aprendendo com outros desenvolvedores e aprimorando seu código em um ambiente real.
- **Crie seus próprios projetos:** Desenvolva aplicações web, jogos, ferramentas de desktop ou qualquer outra coisa que te motive.

# PROXIMOS PASSOS

## Explore Ferramentas e Bibliotecas Úteis:

**Utilize um IDE:** Um bom IDE, como PyCharm ou Visual Studio Code, pode te ajudar a escrever código mais rápido e eficientemente, com recursos como autocompletar, depuração e análise de código.

**Domine o Git:** Aprenda a utilizar o Git para controlar versões do seu código, colaborar com outros desenvolvedores e manter um histórico de suas mudanças.

**Explore ferramentas de teste**: Aprenda a escrever testes unitários e de integração para garantir a qualidade e confiabilidade do seu código.

**Lembre-se:**

A comunidade Python é amigável e acolhedora. Não hesite em compartilhar seus conhecimentos com outros desenvolvedores, nem hesite pedir ajuda quando necessário.

**Com persistência e dedicação, você estará no caminho certo para se tornar um desenvolvedor Python altamente capacitado e pronto para os desafios do mercado!**

# LA CASA DE PYTHON

## OBRIGADO POR LER O EBOOK E ATÉ A PRÓXIMA MISSÃO

# LA CASA DE PYTHON

- Feh Lima – pythonist Jr. Estudante de Analise e Desenvolvimento de Sistemas pelo IFRR. Técnico em publicidade e propaganda. Atividade realizada para o bootcamp de Fundamentos para IA do Santander em parceria com a DIO (Digital Inovation One).
- Disponível em
- https://github.com/users/fehlyma5/projects/3/views/1
- Texto Gerado por IA [Gemini em 10/06/2024] Revisado por Wilson Alves professor responsável pela incubadora de negócios do IFRR, e pela Desenvolvedora Senior Kezia Keullen.